John von Neumann Institute for Computing

Aspects of a Parallel Molecular Dynamics Software for Nano-Fluidics

Martin Bernreuther, Martin Buchholz,
Hans-Joachim Bungartz

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 53-60, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Aspects of a Parallel Molecular Dynamics Software for Nano-Fluidics

Martin Bernreuther[1], Martin Buchholz[2], and Hans-Joachim Bungartz[2]

[1] Höchstleistungsrechenzentrum Stuttgart
70550 Stuttgart, Germany
E-mail: bernreuther@hlrs.de

[2] Institut für Informatik, Technische Universität München
85748 Garching, Germany
E-mail: {buchholm, bungartz}@in.tum.de

The simulation of fluid flow on the nano-scale in the field of process engineering involves a large number of relatively small molecules. The systems we are simulating can be modelled using rigid molecular models assembled from sites with non-bonded short-range pair potentials Each of the sites is described by a set of parameters which are required for the calculation of interactions between sites of the same type. For the interaction of unequal sites, mixed parameter sets have to be calculated. This has to be done for each possible pair of sites. We describe an approach to precalculate and store those mixed parameter sets in a stream, which allows efficient access and gives the flexibility to add new site types easily.

Another focus of our work has been on software engineering techniques. Using the adapter design pattern, we achieved a complete decoupling of the physical parts of the simulation (e.g. molecule models and interactions) from the data structures and the parallelisation. This eases the further concurrent development of the software and reduces the complexity of the different modules. It also gives us the opportunity to swap modules in a plug-in like fashion.

Finally, we demonstrate the advantages of a "pair ownership" of processes for the parallelisation which allows the joint calculation of macroscopic values and the forces on molecules.

1 Introduction

Molecular dynamics generally is about solving the molecules' equations of motion[3,1]. To be able to do that, the forces which act upon each molecule have to be calculated in each time step. The calculation of the forces is based on pair potentials. For each combination of different molecules, a different set of parameters is needed for the force calculation. Section 3 describes a method we have developed to handle those parameter sets efficiently.

As we are using only short-range potentials[5], all necessary pairs can be found by iterating for each molecule over all neighbouring molecules (e.g. using Verlet neighbour lists[3] or a linked-cell data structure[4]). Doing this, one has to take care not to calculate some forces twice. Forces result from pairwise potentials. Hence, processing each pair of molecules once guarantees that all necessary calculations are done once only. We found it useful to clearly distinguish between things to be done for molecules and things to be done for pairs of molecules (see Section 4). In our code, this and the use of modular programming (See Section 2) has led to a better design of the data structures and the parallelisation using domain decomposition.

2 Software Engineering Aspects

For molecular dynamics software, good performance is a crucial property. Unfortunately, efficient programming often conflicts with good software engineering. On the basis of our data structure "ParticleContainer" used to store molecules we will explain how we try to resolve this conflict in our program. In our previous implementation, the particle container was not only responsible for storing the molecules but also for calculating the forces which act on the molecules. The reason for this is that only the particle container class knows how to efficiently access molecules and their neighbours. So the container class had to have information about how to calculate forces. There are two problems with this approach:

- The person implementing the data structure shouldn't have to know much about the calculation of forces.
- The program should be capable to handle different molecule models and a variety of intermolecular potentials. This requires a generic data structure.

We use the adapter design pattern[9] shown in Fig. 1 to get rid of both problems. An adapter class is basically just a wrapper class which connects the particle data structure with some class responsible for the pairwise interactions.

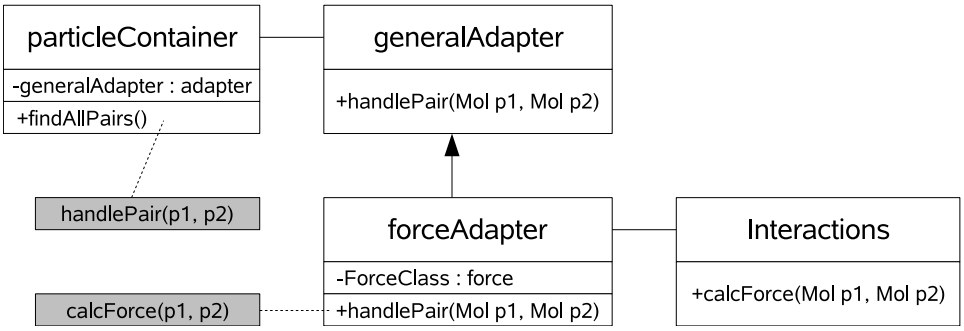


Figure 1. Adapter for pairs

The "ParticleContainer" does not contain any information about how the molecules are modelled and what type of interaction should be used. It just has to provide a method which iterates over all pairs and calls the adapter's "handlePair" method for each pair. Depending e.g. on the material that has to be simulated, a different implementation of the adapter will be used. Typical tasks for an adapter are:

- to call methods which evaluate the potential function for the pair,
- to increase the force on each of the two particles by the calculated value,
- to add contribution of the pair to macroscopic values (e.g. potential, virial,...).

Using this adapter we achieve a high modularity in our program. Different implementations of the ParticleContainer can easily be integrated.

We described how we achieved modularity only on the basis of the data structure "ParticleContainer", but also other parts of the program are designed as flexible modules, expecially the parallelisation and the integrator. This gives us the opportunity to easily develop and test new approaches, e.g. a new parallelisation, and plug them into the program with minimal effort.

3 Force Calculation Based on Parameter Streaming

In the present work molecular dynamics simulations of fluids are realized using rigid molecular models assembled from sites with non-bonded, short range potentials. First of all the Lennard-Jones 12-6 potential

$$U_{ij}^{LJ}\left(r_{ij}\right) = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) \tag{1}$$

covers repulsion and dispersive attraction.

In contrast to the distance $r_{ij} = \left|\vec{r_{ij}}\right| = \left|\vec{r_j} - \vec{r_i}\right|$ between two atoms i and j, the atom diameter σ as well as the energy parameter ϵ are constant. Mixtures of different molecule types, called components, are also supported. The modified Lorentz-Berthelot combining rules

$$\sigma_{AB} = \eta_{AB}\frac{\sigma_A + \sigma_B}{2} \tag{2a}$$

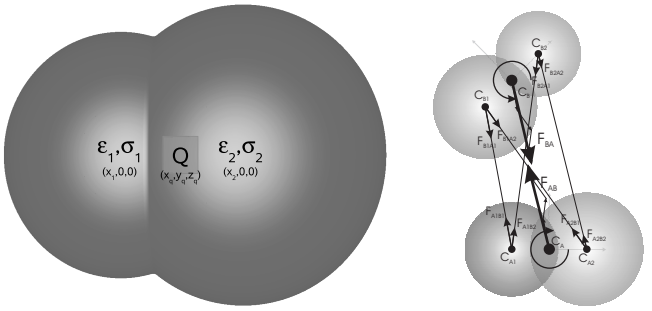
$$\epsilon_{AB} = \xi_{AB}\sqrt{\epsilon_A \epsilon_B} \tag{2b}$$

provide a good starting point to determine the unlike Lennard-Jones parameters for interactions between two different atoms. Whereas η is usually set to one, the binary interaction parameter ξ is a factor close to unity. Assemblies of LJ-centres are well suited for a wide range of anisotropic non-polar molecule types. Quadrupoles or dipoles have to be added to model polar molecules. The corresponding potentials are parameterized with a strength Q or μ for each quadrupole or dipole resp. The force F acting on an atom is directly related to the potential: $\vec{F_{ij}} = -\operatorname{grad} U\left(r_{ij}\right)$.

Regarding multi-centred molecules, the position for each site depends on the position and orientation of the molecule, as well as the local position of the site. The resulting force and torque acting on the molecule are calculated through a vector summation of all the corresponding site-site parts, where each LJ-centre of one molecule interacts with each LJ-centre of the other molecule as well as Dipoles and Quadrupoles interact with each other.

Even though the procedure and implementation works for mixtures of multi-centred molecules with an arbitrary number of sites, the method is demonstrated for the quadrupole two centred Lennard-Jones (2CLJQ) molecule type (cmp. fig. 2(a)). This subclass is suitable to model substances like Nitrogen (N_2), Oxygen (O_2) or Carbon Dioxide (CO_2) and was also successfully applied to binary and ternary mixtures[8]. The potential of and force between two 2CLJQ molecules i and j interacting with each other is given by

$$U_{ij}^{2CLJQ} = 4\sum_{\substack{k=1\\ comp(i)}}^{2}\sum_{\substack{l=1\\ comp(j)}}^{2}\epsilon_{kl}\left(\left(\frac{\sigma_{kl}^2}{r_{kl}^2}\right)^6 - \left(\frac{\sigma_{kl}^2}{r_{kl}^2}\right)^3\right) + \frac{3}{4}\frac{Q_i Q_j}{\left|r_{ij}\right|^5} f(\omega_i, \omega_j) \tag{3a}$$



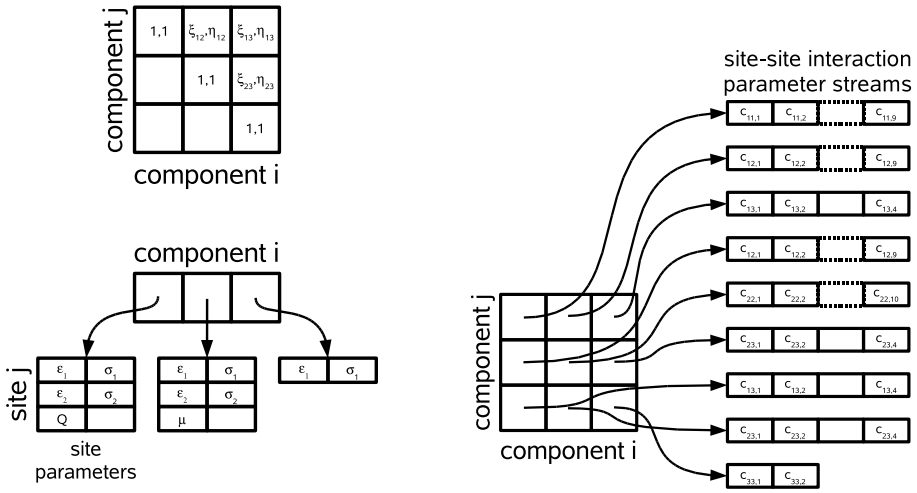
(a) 2CLJQ molecule

(b) Interactions between LJ centres

Figure 2. Exemplification component

$$\vec{F}_{ij}^{2CLJQ} = 24 \sum_{\substack{k=1 \\ comp(i)}}^{2} \sum_{\substack{l=1 \\ comp(j)}}^{2} \epsilon_{kl} \left(2 \left(\frac{\sigma_{kl}^2}{r_{kl}^2} \right)^6 - \left(\frac{\sigma_{kl}^2}{r_{kl}^2} \right)^3 \right) \frac{\vec{r}_{kl}}{r_{kl}^2} + \frac{15}{4} \frac{Q_i Q_j}{r_{ij}^6} f(\omega_i, \omega_j) \frac{\vec{r}_{ij}}{|r_{ij}|} \tag{3b}$$

f is a function related to the orientation ω of the quadrupoles.



(a) storing parameters in arrays

(b) site-site parameters streams

Figure 3. storing parameters

Each component $comp(i)$ has a certain number of sites and each site has a site type dependend number of site parameters. Although equation (2) indicates, that there's also a parameter ξ for each interaction of two LJ centres, a binary interaction parameter ξ will

Runtime per time step ([s])	Intel Core2 (Woodcrest) 2.66 GHz			Intel Itanium2 (Montecito) 1.4 GHz		
Components	Arrays	Streams	Gain	Arrays	Streams	Gain
Argon	0.7141	0.7007	1.88%	2.4744	2.4060	2.77%
Carbon dioxide (CO_2)	1.5962	1.5354	3.81%	5.3185	5.0916	4.27%
Cyclohexane (C_6H_{12})	9.4565	6.6986	29.16%	27.3789	25.7989	5.77%
Air (N_2+O_2+Ar+CO_2)	1.8305	1.5369	16.04%	5.3099	5.0465	4.96%
R227ea	25.6919	18.0649	29.69%	73.7758	68.7716	6.78%

Table 1. Runtime per step comparison of sequential MD simulations of 100000 molecules ($10 mol/l$). (using Intel compiler V10.0 with -fast option)

only be defined for each combination of two different components and used for all the corresponding LJ interactions.

The traditional way to store interaction parameters as well as site parameters is to use multidimensional arrays. A variant is shown in Fig. 3(a), where a two dimensional parameter array is stored for each component. An alternative approach will be presented in the following. We assume a small number of components, which is reasonable for practical experiments. Hence the memory consumption is still acceptable, if we store parameters for each component-component combination. Instead of just saving the site parameters, new site-site interaction parameters are defined, pre-calculated and stored in parameter streams like shown in Fig. 3(b). Note, that even extra parameter streams for the interaction between component i and j as well as j and i exist, where the only difference are the positions and therefore the order of the parameters. As a consequence, the calculation routine doesn't need to handle the molecules in a certain order and saves a conditional statement.

Regarding equations (3), we might define 9 compound parameters and push them in a queue such as | $c_{ij,1}$ | $c_{ij,2}$ | $c_{ij,3}$ | $c_{ij,4}$ | $c_{ij,5}$ | $c_{ij,6}$ | $c_{ij,7}$ | $c_{ij,8}$ | $c_{ij,9}$ | with

$$c_{ij,4k+2l-5} := 4\xi_{ij}\sqrt{\epsilon_k \epsilon_l} \text{ for } (k,l) = (1,1),(1,2),(2,1),(2,2) \tag{4a}$$

$$c_{ij,4k+2l-4} := (0.5 \cdot (\sigma_k + \sigma_l))^2 \text{ for } (k,l) = (1,1),(1,2),(2,1),(2,2) \tag{4b}$$

$$c_{ij,9} := 0.75 \cdot Q_i \cdot Q_j \tag{4c}$$

During the calculation of a molecule-molecule interaction these parameters are read and used in the same order again. Compared to the two times five 2CLJQ site and the interaction parameters, the number of parameters slightly decreased here. Since calculations of equation (4) are only performed once, the overall number of floating point operations will be reduced. Especially the savings of numerous square root calculations is advantageous here. In general however, there might be also a trade of between the number of parameters to be stored resulting in memory usage and the number of floating point operations to be saved, especially if site parameters are used multiple times in various combinations. The choice of optimal streaming parameters might be machine dependent then. An advantage of the stream concept is its flexibility to add new site types. Instead of changing the data structures, the stream will be just extended to include also the new parameter sets.

The implementation uses two functions sketched in listing, which are build up in a symmetric way: the initialization of the streams as a producer and the calculation as a consumer.

Listing 1. Initialization of a stream

```
for {k=0;k<2;++k}
 for {l=0;l<2;++l} {
  eps4 = 4.*xi*sqrt(eps[k]*eps[l]);
  strmij << eps4;
  sig2 = 0.5*(sigma[k]+sigma[l]);
  sig2* = sig2;
  strmij << sig2;
#ifndef NDEBUG
  marker = -1
  strmij << marker;

#endif

 }
 strmij << 0.75*Qi*Qj;
//
//
```

Listing 2. Calculation using a stream

```
for{k=0;k<2;++k}
 for{l=0;l<2;++l}{

  strmij >> eps4;


  strmij >> sig2;
#ifndef NDEBUG

  strmij >> marker;
  assert(marker!=-1);
#endif
// calculation with eps4 and sig2
 }
 strmij >> q075;
// calculation with q075
 assert(strmij.endofstream());
```

It's important that these functions match. To assure the consistency, markers will be added and the end of the stream will be checked if compiled in debug mode, where *NDEBUG* is not set.

Unlike common MPI programs, where only one source is written for a program, that will act as a sender or matching receiver at one point there are two separate functions and source code here. Due to performance reasons any unnecessary conditional statements have to be avoided and there's a need for an optimized calculation routine not carrying the initialization routine inside.

Runtime measurements of sequential MD simulations using a version of the program based on the array and the streaming data structure described above show the superiority of the latter (see Table 1). Modelled with a single LJ-centre, Argon as a small molecule doesn't benefit much opposed to Cyclohexane carrying six LJ centres and a quadrupole and the larger R227ea with its ten LJ-centres, a quadrupole and a dipole.

4 Parallelisation

The major parallelisation strategies for molecular dynamics are force decomposition and spatial decomposition[2]. We are focussing on distributed memory machines with a large number of processors. For such platforms, spatial decomposition has a better performance than domain decomposition[6]. Our implementation uses MPI for the parallelisation.

Pairs for which the two involved particles are owned by different processors require special treatment. Either one process does the calculations and sends the results to the neighbouring process, or both processes do the calculations. The communication costs do not depend on the number of sites in each molecule, as only force and torque have to be transferred. For the calculation of the force between two molecules with n sites, the costs are $\mathcal{O}(n^2)$. Thus, for big molecules, additional communication is preferred to additional computations. But in our applications, only very small molecules with few sites are simulated. That is why we have decided to calculate boundary pairs twice.

The contribution of each pair has to be added to the macroscopic values. This contribution must only be added once per pair. We have introduced ownership for pairs to guarantee this. That means that not only molecules but also pairs are assigned to processes. Each

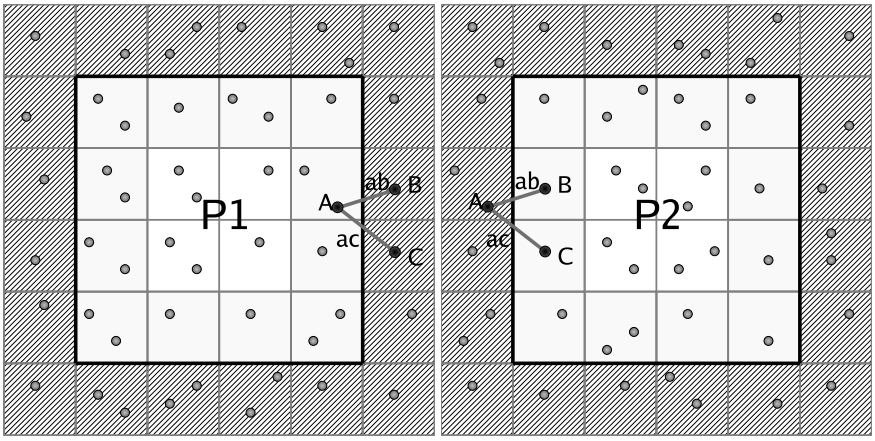


Figure 4. illustration of pairs which cross a process boundary

molecule and each pair belongs to exactly one process. Other molecules and pairs in the boundary region are considered to be copies. While the ownership for molecules is obvious as the processes have distinct spatial domains, the ownership for pairs is more complicated. To decide which process owns a pair, we use a global order of the molecules. A pair belongs to the process who owns the "first" of the two molecules. In our linked cell data structure, we use the index of the cell in which a molecule is stored to define the global order. As molecules which belong to different processes are certainly in different cells, this is sufficient for defining a global ordering. But also any other order criteria (e.g. the id of the particles) would do as well.

Figure 4 shows two neighbouring processes. The grey boundary stripes are the halo regions which contain copies of molecules from neighbouring processes. Particle A (belonging to process $P1$) has the neighbouring particles B and C on process $P2$. The cell of particle C has the lowest index, particle A's cell has a higher index and particle B's cell the highest of the three cells. Hence, the pair ab belongs to process $P1$ and pair ac belongs to process $P2$.

5 Performance and Conclusions

We tested the code on up to 16 nodes of a Linux Cluster. Each node has 8 GB RAM and four Opteron 850 processors with 2.4 GHz. The nodes are connected via an InfiniBand 4x network. First results are shown in Fig. 5. When the problem size is increased from 2.56 million particles to 5.12 million particles, the processing time on a single process increases significantly more than by a factor of two. This is due to insufficient memory on a single machine. Using more processors solves this problem, which explains the superlinear speedup that can be seen in the last row. The runtime comparison with the old version of the code shows only minor differences, hence we have the benefits from a modular and generic program without having to pay for it with reduced efficiency. We plan to conduct more runtime experiments with different configurations to confirm this. Further

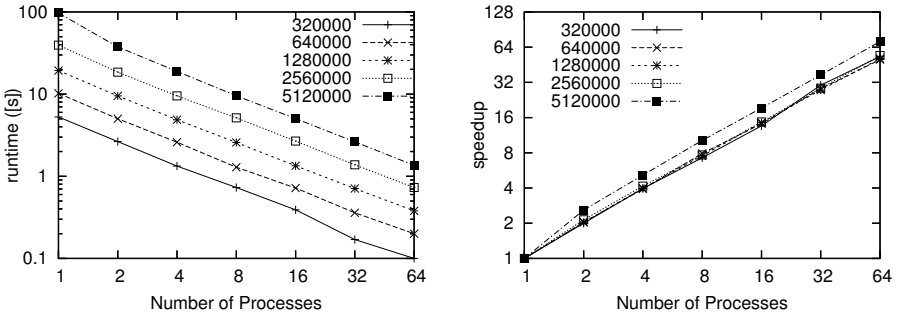


Figure 5. Runtime and speedup for different numbers of particles on up to 64 processors

experiments especially with heterogeneous particle distributions will be done to figure out the importance of load balancing and adaptive data structures for our application.

References

1. D. Fincham, *Parallel computers and molecular simulation*, Molecular Simulation, **1**, 1–45, (1987).
2. S. Plimpton, B. Hendrickson, *Fast Parallel Algorithms for Short-Range Molecular Dynamics*, J. Comp. Phys., **117**, 1–19, (1995).
3. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, (Clarendon Press, Oxford, 1987)
4. M. Griebel, S. Knapek, G. Zumbusch and A. Caglar, *Numerische Simulation in der Molekül̈dynamik*, (Springer, 2004).
5. J. M. Haile, *Molecular Dynamics Simulation - Elementary Methods*, (Wiley, 1997).
6. M. Bernreuther and H. J. Bungartz, *Molecular Simulation of Fluid Flow on a Cluster of Workstations*, in: 18th Symposium Simulationstechnique ASIM 2005 Proceedings, pp. 117–123, (2005).
7. M. Bernreuther and J. Vrabec, *Molecular simulation of fluids with short range potentials*, in: High Performance Computing on Vector Systems: Proceedings of the Second Teraflop Workshop; Stuttgart, March 17–18, 2005, pp. 187–195, (2005)
8. J. Vrabec, J. Stoll and H. Hasse, *Molecular models of unlike interactions in fluid mixtures*, Molecular Simulation, **31**, 215–221, (2005)
9. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, (Addison-Wesley, 1999).